Regis da Silva

Publicado em:

Thu 12 June 2014

←Home

Introdução a Classes e Métodos em Python (básico)

// Tags python classes métodos

Eu não sou a melhor pessoa do mundo para explicar este assunto, mas eu escrevi este post para introduzir um tópico sobre *manipulação de banco de dados em SQLite3 com Python*, porém mais informações sobre classes e métodos podem ser encontradas nos links abaixo. Veja os exemplos em https://github.com/rg3915/pythonDesktopApp.

PS: *Considere a sintaxe para Python 3*.

Segundo a documentação do Python e o video Python para Zumbis, uma **classe** associa dados (**atributos**) e operações (**métodos**) numa só estrutura. Um **objeto** é uma variável cujo tipo é uma classe, ou seja, um **objeto é uma instância** de uma classe.

Na sua sintaxe mais elementar definimos uma classe conforme abaixo:

```
class NomeDaClasse(object):
        pass
```

E um método (função) como:

```
def metodo(args):
        pass
```

onde `args` são argumentos opcionais (parâmetros de entrada). A função `metodo` pode retornar um valor de saída:

```
def metodo(args):
        return args
```

Juntando os dois temos:

```
class NomeDaClasse(object):
        atributo1 = None

        def metodo(args):
                pass
```

`pass` significa que você pode escrever o seu código no lugar. E `atributo1` é um atributo com valor inicial `None` (nada). Poderia ser `atributo1 = 0`, por exemplo.

Importante: Note que para nome de **classes** usamos *PalavrasComeçandoPorMaiúscula* (isso tambem é conhecido como "CamelCase") e para nome de **métodos (funções)** usamos *minúsculas_separadas_por_underscore*. Esta é uma convenção adotada pelos *Pythonistas* segundo o Guia de Estilo **PEP 8** - Style Guide for Python Code escrito por Guido Van Rossum.

Exemplo 1 - Calculadora simples

Existem pelo menos duas formas diferentes de trabalhar com os parâmetros de entrada. Neste exemplo, definiremos o **parâmetro apenas uma vez** com um método especial do Python chamado `__init__`. Segundo João Reis, este método é chamado quando um objeto de uma classe é

instânciado. Este método é útil para fazer qualquer inicialização que você queira com seu objeto, ou seja, ele é o método **"Inicializador"** da instancia.

```
#calculadora.py
class Calculadora(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def soma(self):
        return self.a + self.b

    def subtrai(self):
        return self.a - self.b

    def multiplica(self):
        return self.a * self.b

    def divide(self):
        return self.a / self.b
```

Note que definimos dois parâmetros `a` e `b` (dentro do parênteses). E o `self.a` é um novo campo.

Poderíamos definir

```
def __init__(self, param1, param2):
        self.a = param1
        self.b = param2
```

para não confundir, mas usualmente usamos o mesmo nome tanto no parâmetro quanto no novo campo.

Como dito antes, definimos os valores iniciais apenas uma vez e depois apenas usamos os métodos para calcular os valores.

Podemos rodar o Python no modo modo interativo pelo terminal e importar a classe (veja este video).

```
$ python3
```

```
>>> from calculadora import Calculadora
>>> c = Calculadora(128,2)
>>> print('Soma:', c.soma())
>>> print('Subtração:', c.subtrai())
>>> print('Multiplicação:', c.multiplica())
>>> print('Divisão:', c.divide())
```

`c = Calculadora(128,2)` é uma instância da classe com dois valores iniciais.

O resultado é:

```
>>> Soma: 130
>>> Subtração: 126
>>> Multiplicação: 256
>>> Divisão: 64.0
```

Podemos redefinir os valores iniciais da seguinte forma:

```
>>> c.a = 12
>>> c.b = 42
>>> print c.soma()
```

Resultado:

```
>>> 54
```

Importante: apesar de não fazer parte do escopo deste artigo, mas vejam este video Operadores aritméticos e divisão no Python 2 e Python 3, explicando sobre a diferença no resultado da divisão nas duas versões do Python.

Vejam também este artigo sobre ponto flutuante: Floating Point Arithmetic Issues and Limitations.

Exemplo 2 - Calculadora

Agora faremos uma classe sem valor inicial e com **dois parâmetros** *para todos os métodos*.

```
#calculadora2.py
class Calculadora(object):

    def soma(self, a, b):
        return a + b

    def subtrai(self, a, b):
        return a - b

    def multiplica(self, a, b):
        return a * b

    def divide(self, a, b):
        return a / b
```

Usando o **terminal no modo interativo** façamos:

```
$ python3
>>> from calculadora2 import Calculadora
>>> c = Calculadora()
>>> print('Soma:', c.soma(2,3))
>>> print('Subtração:', c.subtrai(2,10))
>>> print('Multiplicação:', c.multiplica(3,3))
>>> print('Divisão:', c.divide(128,2))
```

A vantagem de colocar os parâmetros em cada método, é que podemos calcular qualquer valor sem ter que instanciar uma nova classe para cada valor diferente.

Exemplo 3 - Classe Pedido

Agora veremos um exemplo que mais se aproxima do que iremos fazer em banco de dados, mas aqui iremos apenas instanciar os objetos e armazená-los em memória numa lista.

Veremos o código na íntegra e depois os

comentários.

```
#user.py
class User(object):

    seq = 0
    objects = []

    def __init__(self, nome, idade):
        self.id = None
        self.nome = nome
        self.idade = idade

    def save(self):
        self.__class__.seq += 1
        self.id = self.__class__.seq
        self.__class__.objects.append(self)

    def __str__(self):
        return self.nome

    def __repr__(self):
        return '<{}: {} - {} -
{}>\n'.format(self.__class__.__name__, self.id,
self.nome, self.idade)

    @classmethod
    def all(cls):
        return cls.objects

if __name__ == '__main__':
    u1 = User('Regis', 35)
    u1.save()
    u2 = User('Fabio', 20)
    u2.save()
    print(User.all())
```

Podemos rodar o Python no modo modo interativo pelo terminal e importar a classe (veja este video).

```
$ python3
>>> from user import User
>>> u1 = User('Regis', 35)
>>> u1.save()
>>> u2 = User('Fabio',20)
>>> u2.save()
>>> print(User.all())
```

Agora os comentários:

Definindo a classe

```
class User(object):
```

Define um atributo que servirá como contador inicial e um atributo `objects` (tupla vazia) que é uma lista de instâncias de `User` que foram salvos (que chamaram o método `save`).

```
seq = 0
objects = []
```

Atribui um valor inicial aos atributos no momento da chamada do construtor.

```
def __init__(self, nome, idade):
```

Inicializando os atributos, `id` começa com `None`, pois a instância foi criada mas ainda não foi salva.

```
self.id = None
self.nome = nome
self.idade = idade
```

Método para salvar os dados ele incrementa o atributo de classe que conta quantas instâncias foram salvas e adiciona a instância na lista de objects.

```
def save(self):
```

`self.__class__` acessa a classe que criou a instância, assim é possível acessar o atributo de `seq`. Aqui poderia ser usado `User.seq`, porém caso `User` fosse herdado, o `seq` seria o de `User` e não da classe filha.

```
self.__class__.seq += 1
self.id = self.__class__.seq
```

Da mesma forma que acessamos `seq`, acessamos objects e é feito um `append` com a instância.

```
self.__class__.objects.append(self)
```

Retorna uma representação do objeto como str, usado em conversões para string. Exemplo: `str(my_user), print my_user`.

```
def __str__(self):
        return self.nome
```

Retorna uma representação do objeto usada para outros objetos. Exemplo: quando é convertida uma lista de user para string.

```
def __repr__(self):
```

`self.__class__.__name__` é a forma de acessar o nome da classe que gerou a instância.

```
return '<{}: {} - {} -
{}>\n'.format(self.__class__.__name__, self.id,
self.nome, self.idade)
```

Class method usado para acessar todas as instâncias salvas (que chamaram o método `save`). Aqui usamos um `@classmethod`, pois faz mais sentido ser um método de classe do que de instância, pois estamos retornando informações da classe e não de uma instância isolada.

```
@classmethod
def all(cls):
```

```
        return cls.objects
```

Demonstração do uso da classe.

```
if __name__ == '__main__':
        u1 = User('Regis', 35)
        u2 = User('Fabio',20)
        print(User.all())
```

Note que nesse `print` a lista está vazia.

```
u1.save()
u2.save()
print(User.all())
```

Após chamar o `save` para as duas instâncias elas são guardadas e o método `User.all()` retorna essa lista.

Agradeço a colaboração de Fabio Cerqueira.

Veja os exemplos em https://github.com/rg3915/pythonDesktopApp.

Mais informações em

Classes Python

A Beginner's Python Tutorial/Classes

The definitive guide on how to use static, class or abstract methods in Python

Python para Zumbis

João Reis

Operadores aritméticos e divisão no Python 2 e Python 3

Floating Point Arithmetic Issues and Limitations